Channel filtering in LArSoft

Gianluca Petrillo, Saba Sehrish, Erica Snider

University of Rochester/Fermilab

LArSoft Coordinators' Meeting, July 28th , 2015

Channel filtering

The channel filter provides information about goodness of each TPC readout channel.
The information hosted so far includes:

bad	channel is dead of irremediably bad
noisy	channel is noisy
non-physical	channel has no actual data

(added by MicroBooNE to describe "wireless" channels)

- currently implemented as a non-configurable class with all the information from all the experiments hard-coded in the constructor
- bug #1083 (!) tracks the issue

Current uses

Very simple to use: instantiate, then query.

```
#include "Filters/ChannelFilter.h"

//...
filter::ChannelFilter chanFilt;

// ...
for(auto & itr : planeIDToHits){
  allhits.resize(itr.second.size());
  allhits.swap(itr.second);

  fDBScan.InitScan(allhits, chanFilt.SetOfBadChannels());
  // ...
}
```

Listing 1: Excerpts from `DBcluster` module

Currently used in:

calibration `recob::Wire` should not be created for bad channels

reconstruction algorithms for track-like clusters check if a gap was due to a bad channel (usually, in the wrong way)

event display learn if channel is bad, draw it accordingly

ChannelFilterProvider

The solution: `ChannelFilterProvider`

Provide access to channel quality information by a service.

This solution was discussed and accepted at LArSoft Architecture meeting on June 24th , 2015:

- as easy as the current one to use in the code
- follows the service provider/framework interface model:
  - `ChannelFilterProvider` provides functionality
  - `ChannelFilterService` provides `ChannelFilterProvider` (from `art`)
- implements interface/implementation model:
  - `ChannelFilterProvider`/`ChannelFilterService` are abstract *interfaces*
  - each experiment must choose a proper implementation

Models

FHiCL file model (implemented in larevt)

This implementation is provided in larevt repository as `SimpleChannelFilter/SimpleChannelFilterService`:

- service FHiCL configuration contains all the channel information
- the content is never updated: all runs have the same lists
- in other words: a glorified `ChannelFilter`

Database model (may be implemented by the experiments)

These are characteristics of a possible DB-based implementation:

- the service configuration contains database connection directions
- the service provider deals with the specific database structure
- the service provider turns queries to the database;
  caching is an implementation detail
- the `art` service triggers content update on every new event

Amended use syntax

Still very simple to use: get the provider, then query.

```
#include "art/Framework/Services/Registry/ServiceHandle.h"
#include "Filters/ChannelFilterService.h"
#include "Filters/ChannelFilterProvider.h"

// ...
filter::ChannelFilterProvider const& chanFilt
  = art::ServiceHandle<filter::ChannelFilterService>()->GetProvider();

// ...

for(auto & itr : planeIDToHits){
  allhits.resize(itr.second.size());
  allhits.swap(itr.second);

  fDBScan.InitScan(allhits, chanFilt.BadChannels());
  // ...
} // for
```

Listing 2: Amended `DBcluster` module code

Reminder on service provider/framework interface

Note the difference with services that are not based on the service provider/framework interface model:

```
filter::ChannelFilterProvider const& chanFilt
  = art::ServiceHandle<filter::ChannelFilterService>()->GetProvider();
```

Listing 3: How to fetch the service provider

1. ask the framework for service/framework interface (`ChannelFilterService`):

   `art::ServiceHandle<filter::ChannelFilterService>()`

2. ask the service/framework interface for the service provider:

   `art::ServiceHandle<filter::ChannelFilterService>()->GetProvider()`

   – the result is a reference to the service provider (`ChannelFilterProvider`), rather than the service/framework interface : `filter::ChannelFilterProvider const& chanFilt`

Source code

Code is in branch `feature/Issue1083` under the following repositories:

larevt in `Filters/`:

- interface headers
- FHiCL-driven service implementation (`SimpleChannelFilter`), with unit test
- `ChannelFilterService` FHiCL configuration files for MicroBooNE, ArgoNeuT, Bo *(the former two should be moved into experiment repositories)*
- updated `ADCFilter` using directly the new service

lbnecode in `lbnecode/lbne/Utilities`:

- `ChannelFilterService` FHiCL configuration files for DUNE, and updated global service configurations

What breaks, what does not

- `ChannelFilter` has been reimplemented to internally use the service:
  - user code does not change...
  - ... but the new service must be configured!!
  - `ChannelFilter` will be eventually deprecated
- in general, `ChannelFilterService` must be configured
- using `ChannelFilterProvider` in place of `ChannelFilter` requires minor, one-to-one changes to user code

Replace `ChannelFilter::...`	... with `ChannelFilterProvider::...`
`BadChannel()`	`isBad()`
`NoisyChannel()`	`isNoisy()`
`SetOfBadChannels()`	`BadChannels()`
`SetOfNoisyChannels()`	`NoisyChannels()`
`GetChannelStatus()`	n/a; use also `isPresent()`, `isGood()`

These changes are documented in `ChannelFilter` class.

Summary

- code is ready in feature branches
- breaking change: experiment configuration need to be updated!
- `ChannelFilter` will be deprecated and eventually removed
  the compiler will tell you
- LArSoft code has not yet been migrated from the deprecated `ChannelFilter` to the service (with one minor exception)
- the name can change upon timely request (cue: speak now)

Backup

Service provider interface

The service provider might follow this interface:

```
class ChannelFilterProvider {
    public:
  using ChannelSet_t = std::set<raw::ChannelID_t>;

  virtual ~ChannelQuality() = default;

  virtual bool isPresent(raw::ChannelID_t channel) const = 0;
  virtual bool isGood   (raw::ChannelID_t channel) const = 0;
  virtual bool isBad    (raw::ChannelID_t channel) const = 0;
  virtual bool isNoisy  (raw::ChannelID_t channel) const = 0;

  virtual ChannelSet_t GoodChannels() const = 0;
  virtual ChannelSet_t BadChannels() const = 0;
  virtual ChannelSet_t NoisyChannels() const = 0;

  virtual bool Update(lariov::IOVTimeStamp const& ts) = 0;

}; // class ChannelFilterProvider
```

Listing 4: `filter::ChannelFilterProvider` *interface*

The `art` service would just return the service provider.

Additional features

Optional features that can be implemented on demand:

1. legacy `ChannelFilter` class reproducing the old behaviour (it will still require the new service to be configured)
2. iterators to channel IDs with specific quality (e.g. good)
3. iterators to channel IDs with custom quality
4. iterators to `raw::RawDigit` (as for channel IDs)
5. interface extension to get channel quality as map of bits
6. ...

None of these have been considered worth implementing immediately.

Additional bit-based interface

```
class filter::ChannelQuality {
    public:
  // the stuff above, plus:

  constexpr size_t NBits = 32;
  using ChannelBits_t = std::bitset<NBits>;

  typedef enum {
    cqNonPhysical,              ///< no wire connected to the channel
    cqDead,                     ///< dead channel
    cqNoisy,                    ///< noisy channel

    cqCustomQualityStart = 16U ///< from this on: experiment-specific
  } ChannelQuality_t;

  virtual ChannelBits_t ChannelStatus
    (raw::ChannelID_t channel) const = 0;
  virtual bool isChannel
    (raw::ChannelID_t channel, ChannelBits_t mask) const;

}; // class filter::ChannelQuality
```

Listing 5: Additional (optional) interface for bit-based quality